Assignment 2 KNN Fundamentals of Machine Learning

2025-06-10

#Assignment 2 - Fundamentals of Machine Learning

##by David Wilkinson

#Load Data and Libraries

#load libraries  
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(ISLR)  
library(class)

#load universal bank data  
library(readr)  
ub.df <- read.csv("C:\\Users\\david\\OneDrive\\Documents\\Kent State University MSBA\\Fundamentals of Machine Learning June 2025\\UniversalBank.csv")  
ub.df <- data.frame(ub.df)

#Data Preparation

#performing one-hot encoding on "Education" field  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(fastDummies)  
encoded.ub.df <- dummy\_cols(ub.df, select\_columns = "Education", remove\_first\_dummy = FALSE, remove\_selected\_columns = TRUE)  
#remove "ZIP.Code and "ID" columns  
encoded.ub.df <- select(encoded.ub.df,-one\_of("ZIP.Code", "ID"))  
#move "Personal.Loan" column to the end of the table  
library(dplyr)  
final.ub.df <- relocate(encoded.ub.df, Personal.Loan, .after = Education\_3)  
head(final.ub.df)

## Age Experience Income Family CCAvg Mortgage Securities.Account CD.Account  
## 1 25 1 49 4 1.6 0 1 0  
## 2 45 19 34 3 1.5 0 1 0  
## 3 39 15 11 1 1.0 0 0 0  
## 4 35 9 100 1 2.7 0 0 0  
## 5 35 8 45 4 1.0 0 0 0  
## 6 37 13 29 4 0.4 155 0 0  
## Online CreditCard Education\_1 Education\_2 Education\_3 Personal.Loan  
## 1 0 0 1 0 0 0  
## 2 0 0 1 0 0 0  
## 3 0 0 1 0 0 0  
## 4 0 0 0 1 0 0  
## 5 0 1 0 1 0 0  
## 6 1 0 0 1 0 0

final.ub.df$Personal.Loan <- factor(final.ub.df$Personal.Loan)

#Data Partitioning

set.seed(20)  
#partition 60% of data based on "Personal.Loan" column  
data\_partition.ub <- createDataPartition(final.ub.df$Personal.Loan, p = 0.6, list = FALSE)  
#assign 60% to training variable  
training\_index <- final.ub.df[data\_partition.ub ,]  
#assign remaining 40% to validation variable  
validation\_index <- final.ub.df[-data\_partition.ub ,]

#Create New Record

#make data frame with new record  
new\_record.ub.df <- data.frame(Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2,   
 Education\_1 = 0, Education\_2 = 1, Education\_3 = 0, Mortgage = 0,   
 Securities.Account = 0, CD.Account = 0, Online = 1, CreditCard = 1)  
#append "Personal.Loan" value to last column of new record, with value "Unknown"  
new\_record.ub.df <- cbind(new\_record.ub.df, Personal.Loan = "Unknown")

#Normalize the Data

#back up original training and validation sets to new variables to be manipulated   
training\_index\_normalized <- training\_index  
validation\_index\_normalized <- validation\_index  
  
#normalize training data using z-scores and assign to new variable  
normalized\_values <- preProcess(training\_index[, 1:13], method = c("center","scale"))  
  
#assign normalized values from first 13 columns of training data to new variable  
training\_index\_normalized[, 1:13] <- predict(normalized\_values, training\_index[, 1:13])  
#assign normalized values from first 13 columns of validation data to new variable  
validation\_index\_normalized[, 1:13] <- predict(normalized\_values, validation\_index[, 1:13])  
#verify output using head function  
head(training\_index\_normalized)

## Age Experience Income Family CCAvg Mortgage  
## 3 -0.5712785 -0.4658652 -1.37006607 -1.2097300 -0.5376169 -0.554696  
## 4 -0.9204886 -0.9897344 0.56657173 -1.2097300 0.4432638 -0.554696  
## 5 -0.9204886 -1.0770460 -0.63022691 1.3813148 -0.5376169 -0.554696  
## 6 -0.7458836 -0.6404883 -0.97838651 1.3813148 -0.8838100 1.007071  
## 7 0.6509566 0.5818732 -0.04270758 -0.3460484 -0.2491225 -0.554696  
## 8 0.3890491 0.3199386 -1.13070634 -1.2097300 -0.9415089 -0.554696  
## Securities.Account CD.Account Online CreditCard Education\_1 Education\_2  
## 3 -0.3412447 -0.2563151 -1.2027052 -0.6395122 1.1661444 -0.6245367  
## 4 -0.3412447 -0.2563151 -1.2027052 -0.6395122 -0.8572409 1.6006533  
## 5 -0.3412447 -0.2563151 -1.2027052 1.5631705 -0.8572409 1.6006533  
## 6 -0.3412447 -0.2563151 0.8311818 -0.6395122 -0.8572409 1.6006533  
## 7 -0.3412447 -0.2563151 0.8311818 -0.6395122 -0.8572409 1.6006533  
## 8 -0.3412447 -0.2563151 -1.2027052 1.5631705 -0.8572409 -0.6245367  
## Education\_3 Personal.Loan  
## 3 -0.6477981 0  
## 4 -0.6477981 0  
## 5 -0.6477981 0  
## 6 -0.6477981 0  
## 7 -0.6477981 0  
## 8 1.5431763 0

head(validation\_index\_normalized)

## Age Experience Income Family CCAvg Mortgage  
## 1 -1.79351374 -1.6882267 -0.5431870 1.3813148 -0.1914237 -0.5546960  
## 2 -0.04746347 -0.1166191 -0.8695866 0.5176332 -0.2491225 -0.5546960  
## 9 -0.92048860 -0.9024229 0.1531322 0.5176332 -0.7684123 0.4931993  
## 13 0.21444407 0.2326270 0.8712114 -0.3460484 1.0779513 -0.5546960  
## 14 1.17477172 1.0184308 -0.7390268 1.3813148 0.3278661 -0.5546960  
## 16 1.26207424 0.8438078 -1.1307063 -1.2097300 -0.2491225 -0.5546960  
## Securities.Account CD.Account Online CreditCard Education\_1 Education\_2  
## 1 2.9294715 -0.2563151 -1.2027052 -0.6395122 1.1661444 -0.6245367  
## 2 2.9294715 -0.2563151 -1.2027052 -0.6395122 1.1661444 -0.6245367  
## 9 -0.3412447 -0.2563151 0.8311818 -0.6395122 -0.8572409 1.6006533  
## 13 2.9294715 -0.2563151 -1.2027052 -0.6395122 -0.8572409 -0.6245367  
## 14 -0.3412447 -0.2563151 0.8311818 -0.6395122 -0.8572409 1.6006533  
## 16 -0.3412447 -0.2563151 0.8311818 1.5631705 -0.8572409 -0.6245367  
## Education\_3 Personal.Loan  
## 1 -0.6477981 0  
## 2 -0.6477981 0  
## 9 -0.6477981 0  
## 13 1.5431763 0  
## 14 -0.6477981 0  
## 16 1.5431763 0

#Perform KNN Using FNN Package

library(FNN)

##   
## Attaching package: 'FNN'

## The following objects are masked from 'package:class':  
##   
## knn, knn.cv

#use knn to predict validation set using k=1  
nn <- knn(train = training\_index\_normalized[, 1:13], test = validation\_index\_normalized[,1:13],   
 cl = training\_index\_normalized[, 14], k = 1, prob = TRUE)  
  
#print off row names of neighbors by distance with k=1  
row.names(training\_index)[attr(nn, "nn.index")]

## [1] "3810" "3666" "4113" "4721" "4172" "4775" "840" "780" "477" "4890"  
## [11] "3276" "3839" "4504" "4588" "494" "2417" "1812" "4953" "2614" "4782"  
## [21] "4133" "4170" "2084" "3185" "185" "3102" "1597" "4134" "331" "1764"  
## [31] "2936" "4511" "3372" "698" "2446" "3331" "4810" "334" "3634" "3979"  
## [41] "4967" "4327" "1357" "1228" "2852" "2314" "4679" "4317" "1775" "4172"  
## [51] "4684" "1240" "582" "3166" "1460" "4152" "3040" "4901" "988" "3909"  
## [61] "2493" "4326" "1065" "3990" "2741" "2305" "1533" "4170" "4933" "2579"  
## [71] "196" "4041" "3199" "2924" "808" "1331" "603" "4597" "29" "688"   
## [81] "2111" "3756" "1174" "4152" "2123" "1251" "3485" "2221" "2252" "4177"  
## [91] "4420" "2126" "2079" "3562" "4985" "4382" "4222" "3683" "89" "1063"  
## [101] "3188" "1603" "293" "798" "3825" "3937" "2166" "419" "169" "4868"  
## [111] "3438" "3511" "2086" "4809" "3597" "2575" "3595" "1418" "200" "3434"  
## [121] "2192" "2873" "3926" "4851" "791" "3535" "3766" "3327" "281" "1689"  
## [131] "568" "649" "3782" "1405" "1163" "1985" "4324" "1350" "3209" "1775"  
## [141] "1349" "2583" "3807" "1221" "4416" "2966" "4880" "2723" "21" "2387"  
## [151] "1554" "3894" "3164" "4775" "2352" "3354" "3894" "203" "2284" "2039"  
## [161] "4854" "3820" "1865" "4262" "4630" "2497" "1011" "2798" "4131" "3654"  
## [171] "331" "46" "2094" "835" "631" "2729" "2897" "4918" "2873" "1917"  
## [181] "3433" "3860" "4689" "1263" "1228" "3531" "507" "478" "1531" "60"   
## [191] "1638" "2600" "3974" "1263" "3228" "2449" "3726" "999" "2393" "2777"  
## [201] "3813" "3909" "4092" "4645" "3030" "1282" "2646" "1201" "1221" "1231"  
## [211] "4859" "143" "4631" "250" "4439" "2620" "3257" "3737" "1570" "4743"  
## [221] "4748" "4070" "2490" "2408" "3492" "2063" "4366" "1767" "3457" "2912"  
## [231] "1994" "173" "3811" "2089" "1520" "3872" "3167" "4948" "1347" "2889"  
## [241] "4412" "1114" "576" "700" "3139" "2633" "3440" "2581" "800" "3922"  
## [251] "1301" "552" "4113" "4223" "1798" "1432" "4431" "3191" "4794" "3738"  
## [261] "4222" "853" "3586" "4217" "4775" "4622" "1550" "2380" "3924" "3809"  
## [271] "3634" "4820" "2347" "1684" "3593" "507" "2228" "3667" "2075" "4215"  
## [281] "2447" "1823" "3032" "4610" "2486" "1604" "3381" "2336" "4138" "125"   
## [291] "2147" "1803" "1775" "4257" "4823" "4004" "2125" "164" "1903" "4539"  
## [301] "3760" "4816" "3419" "3375" "982" "2432" "706" "3204" "101" "2384"  
## [311] "594" "2585" "1891" "3595" "3440" "4085" "2607" "261" "3063" "540"   
## [321] "4053" "2085" "3180" "567" "737" "289" "2610" "2524" "4152" "2723"  
## [331] "2493" "392" "1134" "2729" "225" "2623" "860" "3863" "2988" "4486"  
## [341] "2245" "1199" "3553" "155" "1725" "908" "1607" "2480" "562" "427"   
## [351] "1179" "4950" "3048" "3863" "1576" "1466" "4046" "4357" "1038" "1381"  
## [361] "4395" "3016" "1993" "1029" "2428" "3737" "2093" "1583" "2309" "1893"  
## [371] "1363" "548" "4422" "956" "4098" "2974" "559" "1438" "3641" "527"   
## [381] "3179" "1953" "2224" "3447" "2448" "3195" "2883" "4162" "1363" "2082"  
## [391] "266" "4528" "1185" "2746" "2639" "1857" "1990" "4782" "2351" "117"   
## [401] "47" "1305" "3438" "4426" "1551" "4145" "1158" "2692" "4314" "4175"  
## [411] "2724" "1342" "3852" "4876" "2896" "549" "1935" "3377" "936" "4141"  
## [421] "4556" "3001" "675" "4199" "725" "808" "177" "2729" "972" "4031"  
## [431] "1625" "3846" "3568" "3754" "1083" "3204" "2007" "4739" "3418" "3088"  
## [441] "1879" "4091" "434" "2023" "2135" "3046" "4372" "1580" "4558" "2014"  
## [451] "427" "4588" "1064" "4407" "4569" "1207" "2731" "331" "825" "2941"  
## [461] "3459" "4441" "1472" "2918" "45" "4446" "1466" "4282" "1222" "824"   
## [471] "952" "1849" "4329" "825" "4455" "1183" "4553" "705" "1230" "2820"  
## [481] "2616" "3811" "1038" "248" "2951" "4971" "185" "2744" "3167" "164"   
## [491] "3251" "1858" "2158" "112" "4085" "1791" "2339" "3817" "2757" "2492"  
## [501] "3515" "3559" "2175" "3445" "3807" "4921" "2782" "1155" "778" "40"   
## [511] "4144" "2284" "3894" "1096" "1948" "4695" "3965" "3228" "4676" "586"   
## [521] "911" "4570" "2852" "1051" "3607" "2395" "2035" "2240" "3300" "147"   
## [531] "1485" "3459" "875" "4511" "2051" "1266" "795" "2581" "673" "2367"  
## [541] "4546" "3909" "3834" "865" "1291" "494" "4879" "4629" "4878" "4104"  
## [551] "4936" "3441" "1260" "3336" "3501" "2942" "3176" "2348" "1881" "3375"  
## [561] "1018" "1168" "3793" "4082" "638" "1646" "1998" "4816" "2490" "3224"  
## [571] "1531" "4776" "508" "573" "4353" "698" "3001" "4406" "3509" "1418"  
## [581] "2401" "1992" "2400" "4054" "4148" "370" "1612" "1906" "4735" "127"   
## [591] "565" "1995" "1646" "1457" "2987" "407" "4978" "725" "1731" "1478"  
## [601] "2694" "2093" "4013" "2412" "3364" "927" "742" "3742" "4420" "2868"  
## [611] "2147" "4491" "180" "1199" "3508" "613" "3032" "3632" "834" "3717"  
## [621] "1453" "2039" "4328" "1159" "2708" "4907" "2232" "602" "2498" "1432"  
## [631] "960" "3459" "1602" "4782" "2896" "2800" "568" "1915" "993" "4225"  
## [641] "4864" "3253" "321" "4876" "3276" "3492" "1293" "2226" "4687" "2492"  
## [651] "244" "2852" "4498" "3189" "1199" "1242" "1858" "160" "4456" "2819"  
## [661] "1607" "1621" "1112" "3161" "3469" "2462" "1237" "2806" "853" "4597"  
## [671] "4825" "200" "386" "3438" "3885" "1613" "370" "4098" "2475" "3832"  
## [681] "1806" "2126" "4429" "2282" "4366" "4026" "4257" "1856" "508" "3997"  
## [691] "1655" "2212" "3398" "1703" "997" "4061" "673" "3592" "638" "4628"  
## [701] "1537" "2298" "2139" "4124" "2823" "402" "1316" "399" "4886" "4689"  
## [711] "1434" "2177" "1275" "4620" "2630" "2261" "700" "2046" "3909" "3595"  
## [721] "3690" "2305" "3722" "1583" "1673" "4152" "1566" "1999" "4696" "4"   
## [731] "2315" "185" "4165" "825" "1074" "2338" "1737" "3514" "2903" "608"   
## [741] "4484" "1108" "3148" "2581" "4505" "3653" "1018" "4124" "4455" "2205"  
## [751] "1232" "698" "3568" "3691" "3973" "4774" "4572" "178" "3744" "2107"  
## [761] "3782" "808" "4888" "3967" "2608" "3604" "440" "4372" "2900" "4407"  
## [771] "3666" "2471" "1684" "1713" "2922" "1269" "3156" "2493" "526" "289"   
## [781] "4996" "4144" "1070" "4437" "1635" "1879" "4160" "4638" "1725" "1455"  
## [791] "3361" "4272" "613" "2130" "4092" "3966" "1092" "1699" "1301" "2311"  
## [801] "3584" "3905" "3314" "688" "865" "3469" "2733" "4716" "2919" "461"   
## [811] "498" "881" "456" "2389" "451" "4787" "1725" "3333" "3762" "4441"  
## [821] "4843" "3571" "3634" "3807" "1438" "1801" "2239" "4428" "1407" "3333"  
## [831] "3255" "293" "4218" "1310" "2395" "3040" "4990" "1565" "252" "576"   
## [841] "1737" "1038" "4416" "655" "47" "2919" "10" "890" "875" "2130"  
## [851] "720" "4261" "4664" "2377" "4092" "865" "1964" "4541" "2476" "4257"  
## [861] "3953" "1520" "3322" "84" "3513" "47" "3855" "3133" "1380" "3820"  
## [871] "4871" "4957" "2554" "1839" "982" "3560" "1442" "2951" "2645" "1121"  
## [881] "3606" "1404" "2799" "2746" "1987" "4170" "568" "2549" "1922" "1505"  
## [891] "3210" "1955" "4578" "4854" "1936" "2513" "1155" "3180" "3822" "3585"  
## [901] "1813" "1693" "4684" "2482" "3011" "3785" "4692" "1494" "4773" "875"   
## [911] "4278" "703" "3514" "2305" "2157" "194" "4716" "2248" "4868" "713"   
## [921] "4647" "2753" "2959" "4588" "2339" "3743" "3757" "4510" "3023" "1544"  
## [931] "4764" "475" "4864" "1168" "3445" "997" "2951" "3922" "3744" "1849"  
## [941] "4420" "3688" "3427" "3290" "323" "1640" "3076" "3699" "4780" "3489"  
## [951] "4816" "400" "2196" "142" "593" "4912" "1178" "2429" "3667" "2913"  
## [961] "2371" "3503" "4532" "3093" "1846" "863" "2025" "4381" "3438" "2607"  
## [971] "4783" "2474" "1468" "374" "315" "1765" "3621" "4765" "2660" "756"   
## [981] "1178" "3139" "4510" "1258" "2345" "4323" "3393" "3528" "836" "4990"  
## [991] "4692" "2082" "1638" "1993" "3607" "1366" "1306" "4382" "2896" "1239"  
## [1001] "2039" "3489" "2623" "4310" "1627" "3661" "3666" "2328" "3257" "4324"  
## [1011] "1906" "4416" "1805" "2228" "2603" "2826" "4516" "3057" "1095" "1758"  
## [1021] "3909" "3764" "4873" "3820" "3268" "814" "4473" "2535" "331" "753"   
## [1031] "172" "4177" "2407" "4894" "2492" "4104" "226" "1077" "37" "4806"  
## [1041] "2281" "4386" "2938" "115" "81" "985" "4413" "4549" "4631" "2415"  
## [1051] "843" "3361" "2694" "1934" "4603" "1418" "714" "691" "254" "4341"  
## [1061] "675" "3461" "1251" "4504" "523" "2566" "2492" "4422" "2156" "2947"  
## [1071] "1879" "1596" "2267" "1953" "2726" "999" "714" "3589" "3130" "4687"  
## [1081] "2107" "2372" "2387" "1005" "2786" "3223" "706" "2369" "4939" "2822"  
## [1091] "3808" "2430" "4257" "1699" "4980" "2201" "1388" "1123" "1329" "1551"  
## [1101] "3220" "1231" "1990" "2887" "2245" "2584" "4223" "899" "4489" "417"   
## [1111] "1261" "2973" "1598" "1912" "3199" "3021" "1717" "586" "252" "1495"  
## [1121] "1516" "4835" "4561" "1432" "424" "1453" "4601" "404" "1778" "4329"  
## [1131] "2740" "2377" "2627" "2753" "933" "1751" "4135" "2591" "3762" "4505"  
## [1141] "3139" "1382" "47" "2927" "4963" "3675" "935" "1629" "3374" "834"   
## [1151] "2963" "2428" "43" "3265" "400" "1266" "1937" "875" "2872" "2489"  
## [1161] "2013" "419" "4706" "3255" "4138" "2822" "3457" "3394" "4053" "3186"  
## [1171] "649" "1130" "3985" "125" "2339" "4448" "2947" "3261" "4102" "1382"  
## [1181] "1146" "1451" "2680" "3807" "2065" "2761" "701" "4699" "41" "867"   
## [1191] "1064" "1323" "4662" "3702" "4162" "1358" "1124" "4547" "1845" "3227"  
## [1201] "3654" "3227" "4643" "439" "730" "2289" "3929" "1515" "3874" "618"   
## [1211] "2013" "2938" "67" "2496" "2675" "1116" "579" "2152" "1282" "1356"  
## [1221] "2887" "4876" "4013" "4013" "737" "3030" "2562" "2149" "3464" "2390"  
## [1231] "3180" "2351" "4507" "2014" "4544" "4183" "3154" "3084" "2366" "4122"  
## [1241] "298" "3584" "826" "1077" "714" "2367" "4809" "3914" "3344" "437"   
## [1251] "885" "157" "1019" "4896" "1531" "1893" "1897" "3870" "997" "48"   
## [1261] "440" "2666" "609" "2536" "655" "2553" "3426" "1640" "2227" "4382"  
## [1271] "400" "321" "3511" "3985" "4879" "4501" "2525" "3883" "859" "113"   
## [1281] "3878" "2377" "4659" "4553" "1751" "602" "3666" "2739" "4805" "4544"  
## [1291] "1789" "2063" "3204" "619" "3058" "1287" "1640" "1005" "714" "4528"  
## [1301] "4561" "2623" "1356" "1526" "2203" "3812" "2094" "2398" "797" "3713"  
## [1311] "4122" "4731" "2828" "1941" "3535" "2107" "698" "991" "4175" "3948"  
## [1321] "2197" "4122" "79" "917" "4374" "2693" "433" "4985" "3637" "4549"  
## [1331] "117" "3810" "787" "2913" "2692" "1172" "4044" "1778" "3918" "4382"  
## [1341] "4765" "2873" "1457" "36" "3598" "3865" "3422" "1484" "4357" "3194"  
## [1351] "1845" "4736" "2051" "4890" "4918" "1557" "3203" "1903" "4134" "4511"  
## [1361] "4420" "2407" "2740" "2993" "4303" "1011" "1363" "1057" "1401" "2868"  
## [1371] "4995" "2989" "86" "4301" "3880" "458" "1973" "4107" "3075" "795"   
## [1381] "4618" "867" "4412" "2558" "2942" "2664" "2157" "1468" "1719" "1920"  
## [1391] "2384" "94" "3400" "2974" "3224" "1965" "4165" "3592" "790" "1235"  
## [1401] "2932" "2750" "722" "3767" "977" "2737" "4903" "1484" "170" "4521"  
## [1411] "2856" "1422" "4684" "4953" "3421" "2120" "4381" "3621" "2086" "1432"  
## [1421] "102" "3288" "3525" "3141" "1968" "587" "2474" "451" "1158" "4544"  
## [1431] "3375" "3056" "3169" "2497" "2772" "1953" "1237" "2441" "3333" "1979"  
## [1441] "2499" "2790" "2773" "3086" "876" "4737" "368" "348" "931" "263"   
## [1451] "4513" "2958" "3195" "1112" "3194" "3509" "4847" "1342" "4138" "2740"  
## [1461] "1109" "1440" "724" "2297" "4134" "451" "3852" "2339" "1902" "2474"  
## [1471] "3822" "4809" "505" "2638" "1280" "2137" "2474" "3501" "1787" "3161"  
## [1481] "4901" "1427" "1381" "3585" "1964" "1389" "568" "2415" "2046" "3612"  
## [1491] "1902" "1731" "1057" "3046" "2120" "1008" "2374" "4825" "691" "3188"  
## [1501] "1029" "1670" "1628" "4183" "1198" "3503" "596" "3568" "2710" "2355"  
## [1511] "1447" "4054" "4740" "3156" "3461" "814" "4707" "158" "2821" "1949"  
## [1521] "3063" "4234" "753" "4044" "4057" "1383" "239" "4559" "212" "3438"  
## [1531] "3568" "559" "2820" "1261" "4948" "4137" "4226" "1301" "568" "473"   
## [1541] "142" "4407" "605" "2928" "3384" "1979" "1456" "1062" "1069" "4102"  
## [1551] "4127" "1381" "1167" "340" "124" "757" "3466" "2470" "3016" "2353"  
## [1561] "3188" "4342" "1971" "3591" "124" "4974" "4999" "1381" "4822" "1559"  
## [1571] "349" "1612" "888" "4265" "1271" "1199" "3393" "2966" "2553" "1134"  
## [1581] "4526" "2782" "889" "3699" "1335" "46" "815" "4520" "4794" "3489"  
## [1591] "298" "1078" "4161" "2826" "125" "4878" "1994" "1634" "876" "2959"  
## [1601] "149" "2441" "1830" "4780" "400" "2289" "2896" "1816" "2502" "4716"  
## [1611] "4963" "729" "2118" "3813" "3453" "3333" "1051" "2192" "688" "179"   
## [1621] "75" "355" "2166" "3100" "706" "4267" "586" "4699" "543" "2649"  
## [1631] "4317" "4816" "3462" "3399" "1238" "3762" "4510" "716" "3134" "66"   
## [1641] "647" "3581" "967" "3775" "1731" "561" "1260" "977" "402" "1498"  
## [1651] "2799" "3863" "512" "3545" "3947" "1642" "3076" "1657" "4533" "4798"  
## [1661] "3021" "4696" "3953" "3148" "2959" "2941" "3448" "1749" "4512" "3855"  
## [1671] "3314" "3942" "2693" "1695" "4215" "4562" "578" "1040" "866" "2953"  
## [1681] "791" "4293" "1087" "2913" "1115" "4152" "1685" "4821" "3033" "248"   
## [1691] "4350" "3801" "3672" "1596" "1775" "1049" "4002" "3050" "629" "4190"  
## [1701] "2706" "4014" "4522" "1782" "1028" "2554" "836" "1104" "1717" "450"   
## [1711] "2430" "1676" "2326" "1065" "3034" "1042" "4369" "1544" "3881" "2321"  
## [1721] "203" "2039" "1631" "2145" "4117" "3290" "4528" "2535" "2345" "2852"  
## [1731] "2824" "414" "3948" "2948" "2498" "698" "3726" "1677" "4278" "4081"  
## [1741] "2794" "2775" "3813" "3133" "1038" "2438" "835" "3909" "1047" "1832"  
## [1751] "2441" "630" "1859" "1144" "412" "4258" "3931" "4588" "1499" "792"   
## [1761] "2142" "4744" "2873" "4879" "2486" "3399" "439" "2176" "3441" "835"   
## [1771] "2782" "3763" "1914" "1051" "939" "4374" "4856" "2495" "2627" "3287"  
## [1781] "4379" "89" "1612" "2157" "4973" "3493" "2618" "2820" "1228" "1421"  
## [1791] "3622" "1280" "2415" "4243" "3639" "1130" "1898" "2904" "4948" "2959"  
## [1801] "2014" "501" "1888" "3257" "359" "1652" "4618" "147" "4096" "840"   
## [1811] "1944" "2204" "4608" "394" "3261" "2629" "1260" "3878" "3894" "3904"  
## [1821] "419" "1306" "4069" "2601" "1352" "3851" "716" "4384" "4248" "3923"  
## [1831] "1915" "4547" "3299" "3855" "1574" "3571" "3700" "260" "515" "4162"  
## [1841] "4771" "3228" "2704" "2753" "2436" "4162" "1659" "1135" "4107" "3738"  
## [1851] "763" "4138" "4868" "1647" "3164" "4453" "1160" "3922" "3224" "1011"  
## [1861] "4096" "622" "1309" "4996" "4382" "4933" "1498" "3326" "4843" "2018"  
## [1871] "4809" "1995" "760" "877" "3914" "4689" "1778" "825" "4342" "3377"  
## [1881] "1934" "2228" "2799" "1708" "1526" "1069" "4890" "3031" "3953" "2932"  
## [1891] "1775" "2959" "4508" "4013" "4557" "3203" "2553" "3382" "4798" "1056"  
## [1901] "107" "202" "113" "219" "2951" "4825" "4828" "4456" "1413" "854"   
## [1911] "1565" "551" "1537" "1052" "3243" "2394" "1186" "1979" "2086" "2486"  
## [1921] "1775" "4520" "2761" "4876" "2254" "2666" "1587" "4122" "3377" "688"   
## [1931] "3328" "3905" "1391" "3713" "4991" "3544" "46" "873" "368" "149"   
## [1941] "2459" "4957" "2395" "1096" "3878" "1846" "2429" "3053" "458" "1059"  
## [1951] "1849" "4963" "4587" "155" "2328" "1526" "1096" "1955" "2212" "2559"  
## [1961] "4809" "1144" "4013" "1434" "4924" "168" "4822" "747" "4971" "2417"  
## [1971] "2881" "2852" "45" "890" "4441" "3743" "2244" "3782" "865" "2908"  
## [1981] "3433" "2815" "640" "2390" "4913" "4420" "4222" "1990" "750" "2235"  
## [1991] "4352" "3820" "309" "2705" "3800" "177" "177" "4532" "1812" "4014"

#Perform KNN Classification and Prediction

#perform knn classification using "Personal.Loan" as the dependent variable and the remaining items as inputs/features.  
library(caret)  
model.ub <- train(Personal.Loan ~., data = training\_index,  
 method = "knn",  
 preProcess = c("center","scale"),  
 tuneGrid = expand.grid(k = 1),   
 trControl = trainControl(method = "none"))  
model.ub

## k-Nearest Neighbors   
##   
## 3000 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## Pre-processing: centered (13), scaled (13)   
## Resampling: None

#predict "Personal.Loan" field using KNN model created above.  
predict(model.ub, new\_record.ub.df)

## [1] 0  
## Levels: 0 1

If k=1, the new customer would not accept the personal loan, since the predicted value is 0.

#5-Fold Cross-Validation on Training Data to Determine Best K

set.seed(14)  
#perform leave-one-out 5-Fold cross-validation to determine optimal "k" based on training dataset  
trControl <- trainControl(method ="cv", number = 5, allowParallel = TRUE)  
#find best "k" for prediction Personal.Loan from training data   
model.knn.train <- train(Personal.Loan ~.,   
 data = training\_index,   
 method = "knn",  
 preProcess = c("center", "scale"),   
 tuneGrid = expand.grid(k = seq(1, 20, 2)),   
 trControl = trControl)   
model.knn.train

## k-Nearest Neighbors   
##   
## 3000 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## Pre-processing: centered (13), scaled (13)   
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 2399, 2400, 2400, 2401, 2400   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 1 0.9530011 0.7022374  
## 3 0.9533372 0.6785463  
## 5 0.9526649 0.6627425  
## 7 0.9466644 0.6092748  
## 9 0.9469988 0.6046149  
## 11 0.9436672 0.5710520  
## 13 0.9426683 0.5597111  
## 15 0.9423344 0.5563081  
## 17 0.9406666 0.5355441  
## 19 0.9406688 0.5337784  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 3.

The optimal value is “k=3” based on a five-fold cross-validation of the training data. However, this is overly optimistic because it is only based on the training data.

#Estimate K Accuracy Based on Validation Data As Well

# initialize a data frame with two columns: k, and accuracy.  
library(caret)  
accuracy.df <- data.frame(k = seq(1, 14, 1), accuracy = rep(0, 14))  
# compute knn for different k using validation data.  
for(i in 1:14) {  
 knn.pred <- knn(training\_index\_normalized[, 1:13], validation\_index\_normalized[, 1:13],   
 cl = training\_index\_normalized[, 14], k = i)  
 accuracy.df[i, 2] <- confusionMatrix(knn.pred, validation\_index\_normalized[, 14])$overall[1]   
}  
accuracy.df

## k accuracy  
## 1 1 0.9585  
## 2 2 0.9530  
## 3 3 0.9630  
## 4 4 0.9560  
## 5 5 0.9605  
## 6 6 0.9545  
## 7 7 0.9545  
## 8 8 0.9520  
## 9 9 0.9550  
## 10 10 0.9505  
## 11 11 0.9540  
## 12 12 0.9490  
## 13 13 0.9520  
## 14 14 0.9470

#Use New K to predict new record based on Training and Validation Data

model.knn.all <- train(Personal.Loan ~.,   
 data = training\_index,   
 method = "knn",  
 preProcess = c("center", "scale"),   
 tuneGrid = expand.grid(k=3),   
 trControl = trainControl(method ="none"))   
model.knn.all

## k-Nearest Neighbors   
##   
## 3000 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## Pre-processing: centered (13), scaled (13)   
## Resampling: None

predict(model.knn.all, new\_record.ub.df)

## [1] 0  
## Levels: 0 1

When using k=3, the new customer does not accept the loan.

Train\_Predictors<-training\_index[, 1:13]   
Test\_Predictors<-validation\_index[, 1:13]  
  
Train\_labels <-training\_index[,14]   
Test\_labels <-validation\_index[,14]   
  
Predicted\_Test\_labels <-knn(Train\_Predictors,   
 Test\_Predictors,   
 cl=Train\_labels, k=3)  
# Look at the 5 first values of predicted class (i.e., default status) of test set  
head(Predicted\_Test\_labels)

## [1] 0 0 0 1 0 0  
## Levels: 0 1

#Confusion Matrix for Best K (k=3)

library("gmodels")  
CrossTable(x=Test\_labels,y=Predicted\_Test\_labels, prop.chisq = FALSE)

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Row Total |  
## | N / Col Total |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 2000   
##   
##   
## | Predicted\_Test\_labels   
## Test\_labels | 0 | 1 | Row Total |   
## -------------|-----------|-----------|-----------|  
## 0 | 1744 | 64 | 1808 |   
## | 0.965 | 0.035 | 0.904 |   
## | 0.936 | 0.467 | |   
## | 0.872 | 0.032 | |   
## -------------|-----------|-----------|-----------|  
## 1 | 119 | 73 | 192 |   
## | 0.620 | 0.380 | 0.096 |   
## | 0.064 | 0.533 | |   
## | 0.059 | 0.036 | |   
## -------------|-----------|-----------|-----------|  
## Column Total | 1863 | 137 | 2000 |   
## | 0.931 | 0.068 | |   
## -------------|-----------|-----------|-----------|  
##   
##

#Estimating K for 2nd New Record

second\_new\_record <- data.frame(Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education\_1 = 0, Education\_2 = 1, Education\_3 = 0, Mortgage = 0, Securities.Account = 0, CD.Account = 0, Online = 1, CreditCard = 1)  
cbind(second\_new\_record, Personal.Loan ="Unknown")

## Age Experience Income Family CCAvg Education\_1 Education\_2 Education\_3  
## 1 40 10 84 2 2 0 1 0  
## Mortgage Securities.Account CD.Account Online CreditCard Personal.Loan  
## 1 0 0 0 1 1 Unknown

#5 Fold Cross-Validation to Determine Optimal “k” for new record

#perform 5-fold cross-validation to determine optimal "k" for new record  
set.seed(55)  
trControl <- trainControl(method ="cv", number = 5, allowParallel = TRUE)  
#find best "k" for prediction Personal.Loan from training data   
model2.knn <- train(Personal.Loan ~.,   
 data = training\_index,   
 method = "knn",  
 preProcess = c("center", "scale"),   
 tuneGrid = expand.grid(k = seq(1, 20, 2)),   
 trControl = trControl)   
model2.knn

## k-Nearest Neighbors   
##   
## 3000 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## Pre-processing: centered (13), scaled (13)   
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 2399, 2401, 2400, 2401, 2399   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 1 0.9550049 0.7133425  
## 3 0.9590016 0.7209623  
## 5 0.9539988 0.6689399  
## 7 0.9516627 0.6471055  
## 9 0.9483282 0.6120047  
## 11 0.9459988 0.5930424  
## 13 0.9446671 0.5766637  
## 15 0.9443349 0.5722678  
## 17 0.9430015 0.5571219  
## 19 0.9416676 0.5442619  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 3.

#predict k and outcome  
predict(model2.knn, second\_new\_record)

## [1] 0  
## Levels: 0 1

#Verify with Leave-One-Out Cross-Validation to Determine appropriate “k” value.

#perform leave one out cross-validation to determine optimal "k"  
trControl <- trainControl(method ="loocv", number = 5, allowParallel = TRUE)  
#find best "k" for prediction Personal.Loan from training data   
model2.knn <- train(Personal.Loan ~.,   
 data = training\_index,   
 method = "knn",  
 preProcess = c("center", "scale"),   
 tuneGrid = expand.grid(k = seq(1, 20, 2)),   
 trControl = trControl)

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,  
## : There were missing values in resampled performance measures.

model2.knn

## k-Nearest Neighbors   
##   
## 3000 samples  
## 13 predictor  
## 2 classes: '0', '1'   
##   
## Pre-processing: centered (13), scaled (13)   
## Resampling: Leave-One-Out Cross-Validation   
## Summary of sample sizes: 2999, 2999, 2999, 2999, 2999, 2999, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa  
## 1 0.9563333 0   
## 3 0.9576667 0   
## 5 0.9526667 0   
## 7 0.9516667 0   
## 9 0.9503333 0   
## 11 0.9473333 0   
## 13 0.9466667 0   
## 15 0.9450000 0   
## 17 0.9436667 0   
## 19 0.9430000 0   
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 3.

predict(model2.knn, second\_new\_record)

## [1] 0  
## Levels: 0 1

Leave one out cross-validation verifies that k=3 and that the customer will not accept the loan.

#Repartitioning into Training, validation, and test sets.

set.seed(20)  
data\_to\_partion\_3ways <- createDataPartition(final.ub.df$Personal.Loan, p=0.5, list = FALSE)  
Training\_Data <- final.ub.df[data\_to\_partion\_3ways,]  
ValTest\_Data <- final.ub.df[-data\_to\_partion\_3ways,]  
  
ValTest\_Index <- createDataPartition(ValTest\_Data$Personal.Loan, p = 0.6, list = FALSE)  
Val\_Data <- ValTest\_Data[ValTest\_Index,]  
Test\_Data <- ValTest\_Data[-ValTest\_Index,]  
  
summary(Training\_Data$Personal.Loan)

## 0 1   
## 2260 240

summary(Val\_Data$Personal.Loan)

## 0 1   
## 1356 144

summary(Test\_Data$Personal.Loan)

## 0 1   
## 904 96

#Normalize the Data 3 Ways

#back up original training and validation sets to new variables to be manipulated   
training\_normalized <- Training\_Data  
validation\_normalized <- Val\_Data  
test\_normalized <- Test\_Data  
  
  
#normalize training data using z-scores and assign to new variable  
normalized\_values2 <- preProcess(Training\_Data[, 1:13], method = c("center","scale"))  
#assign normalized values from first 13 columns of training data to new variable  
training\_normalized[, 1:13] <- predict(normalized\_values2, Training\_Data[, 1:13])  
#assign normalized values from first 13 columns of validation data to new variable  
validation\_normalized[, 1:13] <- predict(normalized\_values2, Val\_Data[, 1:13])  
#assign normalized values from first 13 columns of test data to new variable  
test\_normalized[, 1:13] <- predict(normalized\_values2, Test\_Data[, 1:13])